Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет телекоммуникаций
и информатики»

Факультет ИВТ

Кафедра вычислительных систем

Курсовая работа

на тему «Обработка последовательной информации»
Вариант 1.4 «Поиск палиндромов в тексте»

Выполнил:
студент гр. ИС-142
Наумов А.А.

Проверил:
преподаватель Кафедры ВС
Фульман В.О.

Новосибирск, 2022

Задание

Разработать программу ***palindrom***, выполняющую поиск всех палиндромов в заданном тексте. Команда ***palindrom*** принимает в качестве аргумента команд- ной строки имя файла, содержащего текст на русском языке. Все найденные палиндромы распечатываются на экране.

Критерии оценки

- Оценка «удовлетворительно»: реализована проверка того, что весь текст входного файла целиком является палиндромом. Не предусмотрено ди- намическое выделение памяти под входные данные.
- Оценка «хорошо»: реализована предварительная обработка текста: из каждого предложения удаляются все знаки препинания. После этого осуществ- ляется проверка каждого предложения на выполнение свойства палиндрома. Обязательно динамическое выделение памяти под входные данные.
- Оценка «отлично»: реализована предварительная обработка текста: из текста удаляются все пробелы и знаки препинания так, чтобы получилось одно большое слово. Для поиска подпалиндромов используется алгоритм, основан- ный на применении *динамического программирования* (http://comp-science.narod.ru/WebPage/p6.html). Обязательно динамическое вы- деление памяти под входные данные.

Указания к выполнению задания

Палиндромом называются слово (потоп, шалаш) или текст (а роза упала налапу Азора), читающиеся одинаково в обоих направлениях.

Запуск программы должен производиться со следующими аргументамикомандной строки:

```
$ palindrom text.txt
```

Программа выполяет поиск всех палиндромов в файле *text.txt* и распечаты- вает результат работы на экране.

Анализ задачи

1. При запуске программы необходимо проверить аргументы командной строки на валидность. Используем для этого функцию **checkArgs**: проверяем, что количество аргументов равно двум (включая имя исполняемого файла), второй аргумент заканчиваются на **.txt**, используя функцию **isSuffix**, которая с конца заданных строк сравнивает их лексикографически с заданным суффиксом (аналог **strcmp** / **sequal**). Если суффиксы различны, возвращается значение **-1** и программа завершается с выводом сообщения об ошибке.

2. Выделяем память под будущий массив слов ***words** функцией **malloc**, в которую подаём размер указателя на символ (**wchar_t***), который в начале равняется константе **SIZE_INCREMENT**, заданной директивой препроцессора **#define**. **SIZE_INCREMENT = 1** для экономии вызовах функции **realloc** (для динамического выделения памяти под массив слов).

3. Теперь необходимо записать данные, которые представлены в текстовом файле как набор

строк из символов алфавита, возможно разделёнными знаками препинания.
Если всё получилось, выполняем данные операции в функции **getInput()**, которая принимает на вход имя входного файла **filename** (**char***) и указатель на счётчик букв **LengthCount** (**int**). Для начала проверим, что файл действительно существует, и, если нет, выведем сообщение об ошибке и завершим программу.

4. При успешном открытии файла с помощью функции **fgetswc** считываем последовательно символов из файла в **buffer.**

5. Полученный **buffer** записываем в новый массив **words.**

6. На каждой итерации цикла **while** со считыванием слов проверяем, не больше ли счётчик слов **lenghtcount** (**int**) чем размер массива слов **size**. В случае, если счётчик слов стал больше или равен размеру массива, перевыделяем под него память функцией **realloc**, увеличивая размер на **SIZE_INCREMENT**. В самом массиве при занесении слова выделяем под него память функцией **malloc**, размер слова находим функцией **strlen** и добавляем 1 для завершающего нуля. Копируем содержимое «чистой» строки **buffer** в **words[wordCount]** функцией **strcpy**. Прибавляем к счётчику **wordCount** единицу и идём дальше по циклу, заполняя массив всеми словами. Закрываем входной файл функцией **fclose**.

7. После успешного считывания слов, выводим массив функцией **printf(“%ls\n”,words)** строкой.

8. Создаем три переменные **wchar_t** формата ***s**(для третьего аргумента в функции деления на токены), *p(для деления на токены) делим весь массив на токены по точке, восклицательному знаку и вопросительному(для проверки на палиндром), ***copy**(для изначального предложения до точки) сразу же выделяем память функцией **malloc**.

9. Далее делаем проверку если ***p** не нулевой, создаем цикл по ***p** и используя, **wcscpy** функцию копируем ***p** предложение до делиметра(**!.?**) в **copy**.

10. Далее воспользуемся функцией удаления знаков препинания **remove_prep**.

11. Все буквы верхнего регистра необходимо преобразовать в нижний регистр, воспользуемся функцией **wtolower**.

12. Необходимо также удалить пробелы в предложении. Используем функцию **remove_ch(p,’ ’);**

13. После данных манипуляций над предложением, оно преобразуется в строчный текст, без пробелов, знаков препинания и имеет буквы нижнего регистра, для проверки на палиндром.

14. После проверки выводим предложение или слово с пометкой что это палиндром и **wcstok**’аем переменную ***p** для дальнейшей проверки текста.

15. После работы программы освобождаем переменную **copy**.

Тестовые данные

Несуществующий входной текстовый файл

```
alexey@DESKTOP-1NTB03J:~/palindrome$ ./palindrome ne.txt
Невозможно открыть файл ne.txt

 Используйте команду:
        palindrome <текстовый документ>

alexey@DESKTOP-1NTB03J:~/palindrome$
```

Неправильно введенный аргумент:

```
alexey@DESKTOP-1NTB03J:~/palindrome$ ./palindrome ne.txt a

 Используйте команду:
        palindrome <текстовый документ>
```

Пустой файл:

```
alexey@DESKTOP-1NTB03J:~/palindrome$ cat net.txt
alexey@DESKTOP-1NTB03J:~/palindrome$ ./palindrome net.txt
alexey@DESKTOP-1NTB03J:~/palindrome$
```

Работа программы:

```
alexey@DESKTOP-1NTB03J:~/palindrome$ cat text.txt
Аргентина манит негра.поп.утро.довод.А роза упала на лапу Азора.Молебен о коне белом.Учуя молоко, я около мяучу.Лёша
на полке клопа нашёл.Я — арка края.Я иду съ мечемъ судия.Муза, ранясь шилом опыта, ты помолишься на разум.Он — верба,
 но / Она — бревно.Уж редко рукою окурок держу.
alexey@DESKTOP-1NTB03J:~/palindrome$ ./palindrome text.txt
Аргентина манит негра.поп.утро.довод.А роза упала на лапу Азора.Молебен о коне белом.Учуя молоко, я около мяучу.Лёша
на полке клопа нашёл.Я — арка края.Я иду съ мечемъ судия.Муза, ранясь шилом опыта, ты помолишься на разум.Он — верба,
Лес.Полоса. Лидер бредил.ко рукою окурок держу.
Аргентина манит негра - это палиндром
поп - это палиндром
довод - это палиндром
А роза упала на лапу Азора - это палиндром
Молебен о коне белом - это палиндром
Учуя молоко, я около мяучу - это палиндром
Лёша на полке клопа нашёл - это палиндром
Я — арка края - это палиндром
Я иду съ мечемъ судия - это палиндром
Муза, ранясь шилом опыта, ты помолишься на разум - это палиндром
Он — верба, но / Она — бревно - это палиндром
Уж редко рукою окурок держу - это палиндром
 Лидер бредил - это палиндром
alexey@DESKTOP-1NTB03J:~/palindrome$
```

Листинг программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "input.h"
#include "palindrome.h"
#include <wchar.h>
#include <locale.h>

int main(int argc, char **argv)
{
    if(checkArgs(argc, argv) == -1) {
        return -1;
    }
    int lengthCount = 0;
    setlocale(LC_ALL, "");
    wchar_t *words = getInput(argv[1], &lengthCount);
    if(words == NULL) {
        printf("Невозможно открыть файл %s\n", argv[1]);
        printHelp();
        return 0;
    }
    wchar_t *copywords = mal-
loc(wcslen(words)*sizeof(wchar_t));
    wcscpy(copywords,words);
    remove_ch(words,' ');
    if(!*words){
        return -1;
    }
    printf("%ls\n", copywords);
    wchar_t *s;
    wchar_t *p=wcstok(copywords,L"!.?",&s);
    wchar_t *copy = malloc(wcslen(s)*sizeof(wchar_t));

    if (p!=NULL)
        {while(p){
        wcscpy(copy,p);
        remove_ch(p,' ');
        remove_prep(p);
        wtolower(p);
        if(isPalindrome(p))
        printf("%ls - это палиндром \n", copy);
        p=wcstok(NULL,L"!.?", &s);
    }
        }
    free(copy);
    return 0;
}
```

input.c

```c
#include "input.h"
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>

#define SIZE_INCREMENT 1

void printHelp() {
    printf("\n Используйте команду:\n");
    printf("\tpalindrome <текстовый документ>\n\n");
}

int isSuffix(char *substring, char *string) {
    int n1 = strlen(substring), n2 = strlen(string);
    if (n1 > n2) {
        return -1;
    }
    for (int i=0; i<n1; i++) {
        if (substring[n1 - i - 1] != string[n2 - i - 1]) {
            return -1;
        }
    }
    return 0;
}

int checkArgs(int argc, char **argv) {
    if(argc != 2 || isSuffix(".txt", argv[1]) != 0) {
        printHelp();
        return -1;
    }
    return 0;
}

wchar_t *getInput(char *filename, int *lengthCount) {
    *lengthCount = 0;
    wchar_t buffer;
    int size = SIZE_INCREMENT;
    wchar_t *words = (wchar_t*)malloc(size * sizeof(wchar_t));
    if(words == NULL) {
        printf("Ошибка при выделении памяти для массива\n");
        free(words);
        return NULL;
    }
    FILE *input = fopen(filename, "r");
    if(input == NULL) {
        free(words);
        return NULL;
    }
    while(1) {
        buffer = fgetwc(input);
        if(feof(input)){
            break;
        }
        if(buffer == '\n'){
            continue;
        }
        if(*lengthCount >= size)  {
```

```
            size += SIZE_INCREMENT;
            words = (wchar_t*) realloc(words, size * sizeof(wchar_t));
        }
        words[*lengthCount]=buffer;
        (*lengthCount)++;
    }
    fclose(input);
    return words;
}
```

input.h

```
1 #pragma once
2 #include <wchar.h>
3
4 void formatString(wchar_t *p);
5 void printHelp();
6 int checkArgs(int argc, char **argv);
7 wchar_t *getInput(char *filename, int *wordCount);
8 int isSuffix(char *substring, char *string);
```

palindrome.c

```
#include "palindrome.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>

wchar_t abc[]=
L"абвгдеёжзийклмнопрстуфхцчшщьыъэюя";
wchar_t ABC[]=
L"АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ";
wchar_t word[256]= { 0 } ;

int isPalindrome(wchar_t *str)
{
    int l = 0;
    int h = wcslen(str) - 1;
    while (h > l)
    {
        if (str[l++] != str[h--])
        {
            return 0;
        }
    }
    return 1;
}

wchar_t *wtolower(wchar_t *str){
    for(int j = 0; j < wcslen(str);j++){
    if (wcschr(ABC,str[j])){
        wchar_t symbol=L'0';
        wcsncpy(&symbol,wcschr(ABC,str[j]),1);
        for(int i=0; i<wcslen(ABC);i++){
            if(ABC[i]==symbol)
            str[j] = abc[i];
        }
    }
    }
    return str;
```

```
}

wchar_t remove_ch(wchar_t *s, wchar_t ch){
    int i,j;
    j=0;
    for(i=0;s[i];i++){
        if(!iswspace(s[i]))
            s[j++] = s[i];
    }
    s[j] = '\0';
    return 0;
}

void remove_prep(wchar_t *s){
    wchar_t *last = s;
    while (*last){
        while(*last && !iswpunct(*last))
                *s++ = *last++;
        while (*last && iswpunct(*last))
        last++;
    }
    *s= '\0';
}
```

palindrome.h

```
#pragma once
#include <wchar.h>

wchar_t remove_ch(wchar_t *s, wchar_t ch);
wchar_t *wtolower(wchar_t *str);
int isPalindrome(wchar_t *str);
void remove_prep(wchar_t *s);
```

Makefile

```
CC+FLAGS = gcc -Wall -MMD

all: palindrome
-include *.d

palindrome: main.o input.o palindrome.o
        $(CC+FLAGS) -o $@ $^

palindrome.o: palindrome.c
        $(CC+FLAGS) -c -o $@ $<

input.o: input.c
        $(CC+FLAGS) -c -o $@ $<

clean:
        rm -rf *.o *.d palindrome
```